DigitalHouse > Coding School

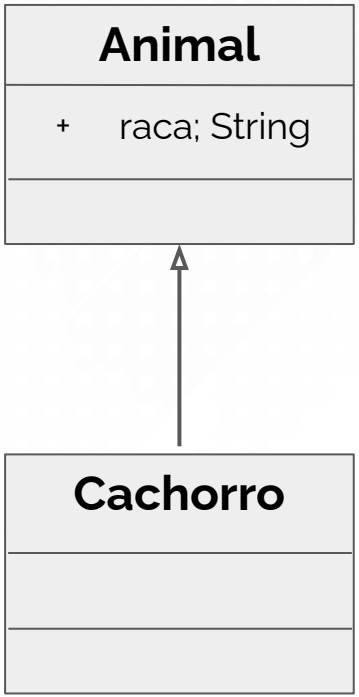
Aula 08

Herança e Classe Abstrata

DigitalHouse >
Coding School

Herança

A herança é um mecanismo da Orientação a Objeto que permite criar novas classes a partir de classes já existentes, aproveitando-se das características existentes na classe a ser estendida.



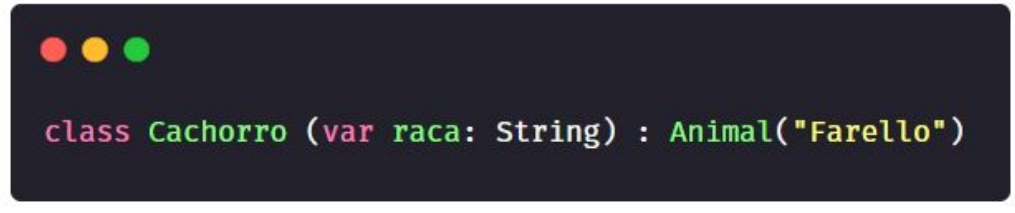
DigitalHouse > Coding School

Para herdar uma classe, é preciso deixar a classe-pai aberta

```
open class Animal (var nome: String)
```

Herança - sintaxe

Implementação da herança



Através dos dois pontos é feita a herança

Herança - sintaxe

Se a classe derivada tiver um construtor primário, a classe base pode (e deve) ser inicializada ali mesmo, usando os parâmetros do construtor primário.

```
open class Base (p: int)

class Derived(p: Int) : Base(p)
```

Override (Sobrescrita)

É utilizado para sobrescrever o comportamento de uma função do pai em uma classe filho.

https://kotlinlang.org/docs/reference/classes.html

Override (Sobrescrita) de funções e Polimorfismo - Exemplo

```
open class Shape {
    open fun draw() { /* ... */ }
    fun fill() { /* ... */ }
}

class Circle : Shape() {
    override fun draw() { /* ... */ }
}
```

Override (Sobrescrita) de funções e Polimorfismo - Exemplo

Uma função marcada como aberto pode ser sobrescrita nas subclasses. Se você deseja proibir a sobrescrita, use a palavra-chave **final** :

```
open class Retangle : Shape() {
    final override fun draw() { /* ... */ }
}
```

Override (Sobrescrita) de propriedades

Substituir propriedades funciona de maneira semelhante a substituir métodos; as propriedades declaradas em uma superclasse que são redeclaradas em uma classe derivada devem ser precedidas de substituição e devem ter um tipo compatível.

```
open class Shape {
    open val vertexCount: Int = 0
}

open class Retangle : Shape() {
    override val vertexCount  = 4
}
```

Super

Para fazer uma referência da classe derivada para a sua classe pai utilizamos a palavra-chave **super.**

```
class Cachorro(nomeAnimal: String) : Animal(nomeAnimal) {

    override fun ehMamifero(): Boolean {
        println("Nome do animal é ${super.nomeAnimal}")
        return super.ehMamifero()
    }
}
```

Modificadores de visibilidade

private - significa visível somente nesta classe (incluindo todos os seus membros);

protected - o mesmo que privado + visível nas subclasses também;

internal - qualquer cliente dentro deste módulo que vê a classe declarante vê seus membros internos;

public - qualquer cliente que vê a classe declarante vê seus membros públicos.

https://kotlinlang.org/docs/reference/visibility-modifiers.html

Exemplo prático de Herança...

Classes Abstratas

As classes abstratas não permitem realizar nenhum tipo de instância, além disso as classes abstratas servem de modelo para as classes derivadas.

Classes Abstratas

"é um tipo de"

Classe Abstrata - Sintaxe

Palavra-chave

Funções Abstratas

As funções abstratas indicam que todas as classes filhas concretas devem reescrever a função abstrata da classe abstrata.

Lembre-se que a função deve também ser abstrata.

Funções Abstratas - Exemplo

Classe abstrata

```
abstract class Mamifero {

    abstract fun racaAnimal(raca: String)
}
```

Classe derivada

```
class Cachorro: Mamifero {

    override fun racaAnimal(raca: String) {
        println("Sobrescrita do método abstrato")
    }
}
```

Exercícios

DigitalHouse >
Coding School